Δομές Δεδομένων-Εργασία 2

Πλαβός Δημοσθένης 3200156

ΜΕΡΟΣ Α

Στο μέρος αυτό, χρησιμοποίησα την ουρά του φροντιστηρίου με ορισμένες παραλλαγές.

Έχω δύο αρχεία, τα MaxPQ.java και MaxPQInterface.java,που το πρώτο κάνει implement το δεύτερο. Οι public μέθοδοι που χρησιμοποιώ είναι οι:

- Void insert(T item): εισάγει ένα αντικείμενο item τύπου T στην ουρά προτεραιότητας.
- T peek(): επιστρέφει τη ρίζα, το αντικείμενο τύπου T που βρίσκεται στην αρχή της ουράς προτεραιότητας.
- T getMax(): επιστρέφει τη ρίζα, αλλά ταυτόχρονα την αφαιρεί από την ουρά.
- void sink(int i): κάνει κατάδυση το στοιχείο που βρίσκεται στο index i της ουράς προτεραιότητας, προκειμένου να πάει στην κατάλληλη θέση.

Επίσης, χρησιμοποιώ την κλάση Comparable, αφού τα αντικείμενα τύπου Disk που θα βάλω στην ουρά, πρέπει να υλοποιούν το interface Comparable<Disk>, και τη μέθοδο compareTo, που υλοποιώ στην κλάση Disk, για τις συγκρίσεις που απαιτούνται όταν κάνω swim και sink.

ΜΕΡΟΣ Β

Στο μέρος Β, υλοποιώ την κλάση Greedy. Η υλοποίησή της γίνεται ως εξής:

Αρχικά, δημιουργώ μία στατική ακέραια μεταβλητή i για να τη χρησιμοποιήσω στην παραγωγή διαφορετικών κάθε φορά id για διαφορετικούς δίσκους με την εντολή i++.

Στη συνέχεια, υλοποιώ τη μέθοδο **isNumeric** η οποία εξετάζει αν μία συμβολοσειρά έχει αριθμητική τιμή, αν μπορεί δηλαδή να γίνει μετατροπή της σε int.

Μετά, υλοποιώ τη μέθοδο **ReadFile(File fl)**, η οποία θα αναγνώσει τα αρχεία που περιέχουν τα μεγέθη των φακέλων και θα επιστρέψει μία λίστα ακεραίων με τα μεγέθη αυτά.

‘ Οσο ,λοιπόν, υπάρχει κείμενο, το διαβάζω γραμμή-γραμμή και περνάω στην μεταβλητή str την συμβολοσειρά που είναι γραμμένη σε κάθε γραμμή. Αν η συμβολοσειρά αυτή δεν έχει τιμή αριθμητική ή έχει τιμή αριθμητική αλλά είναι <0 ή >1000000, πετάω εξαίρεση. Σε αντίθετη περίπτωση, την περνάω μέσα στη λίστα με όνομα arr και την επιστρέφω όταν τελειώσει η ανάγνωση του αρχείου και εφόσον δεν προέκυψε κάποια εξαίρεση.

toDO (List<Integer> arr)

Μετά, υλοποιώ τη μέθοδο *toDO (List<Integer> arr)* η οποία έχει τις βασικές λειτουργίες για την εκτέλεση του αλγορίθμου. Σαν όρισμα, παίρνω τη λίστα με τα μεγέθη των φακέλων, οι οποίοι πρέπει να καταχωρηθούν σε δίσκους.

Δημιουργώ μία ουρά προτεραιότητας με όνομα **pq** η οποία περιέχει αντικείμενα τύπου **Disk**. Στην αρχή, δημιουργώ ένα νέο αντικείμενο τύπου Disk και το περνάω μέσα.

Η προτεραιότητα μέσα στην ουρά προτεραιότητας διαμορφώνεται με βάση τον ελεύθερο χώρο μνήμης που έχει κάθε δίσκος (ο οποίος αρχικοποιείται ως 1000000), οπότε στην κορυφή της ουράς βρίσκεται πάντα ο δίσκος με τον περισσότερο χώρο (μέθοδος getFreeSpace για να επιστραφεί ο χώρος μνήμης). Θέλω, λοιπόν, να «περάσω» τους φακέλους σε δίσκους. Για κάθε φάκελο που διαβάζω, αν ο χώρος μνήμης του δίσκου στην κορυφή της ουράς, είναι μεγαλύτερος του φακέλου, τότε «περνάω» τον φάκελο στον δίσκο και ανανεώνω την τιμή του ελεύθερου χώρου μνήμης του δίσκου (με τη μέθοδο setSize). Αν ο χώρος δεν είναι αρκετός, τότε δημιουργώ νέο δίσκο και περνάω εκεί τον φάκελο.

Χρησιμοποιώ επίσης τη μέθοδο AddFolderToList, έτσι ώστε να καταγράψω ποιος φάκελος ανήκει σε ποιον δίσκο, που το χρειάζομαι για το μήνυμα εκτύπωσης.

Τέλος, σημαντικό είναι ότι κάθε φορά που συμβαίνει μία δημιουργία νέου δίσκου, ή η αλλαγή μεγέθους ενός δίσκου επειδή κάποιος φάκελος εισήχθη σε αυτόν, κάνω sink τον συγκεκριμένο δίσκο(που βρίσκεται πάντα στη ρίζα) προκειμένου να μπει στη θέση της ουράς προτεραιότητας που πρέπει. Εντολή: pq.sink(1).

Ακολουθούν μηνύματα εκτύπωσης για τα οποία χρησιμοποιώ τις μεταβλητές numberOfFolders, FoldersMemory, DisksUsed.

Η μέθοδος toDo επιστρέφει τον αριθμό των δίσκων που χρησιμοποιήθηκαν.

ΜΕΡΟΣ Γ

Στο Μέρος Γ υλοποίω τον αλγόριθμο ταξινόμησης ***Heapsort***.

Έχω τη βασική μέθοδο **HeapSort** και τη βοηθητική **heapify**. Στην heapify συγκρίνω τον πατέρα με τα παιδιά και κάνω swap αν δεν ισχύει ότι η τιμή του πατέρα είναι μικρότερη των παιδιών.

Κάνω εφαρμογή min heap , το οποίο σημαίνει ότι κάθε στοιχείο γονέας πρέπει να είναι μικρότερο από τα παιδία του οπότε γίνονται και οι αντίστοιχες συγκρίσεις

Στην αρχή, το ελάχιστο στοιχείο πρέπει να είναι η ρίζα. Όταν γίνει αυτό, την αφαιρώ από ρίζα, κάνοντας την swap με το τελευταίο στοιχείο του σωρού. Συνεχίζω την ίδια διαδικασία έως ότου το μέγεθος του σωρού φτάσει 1. Έτσι, καταλήγω να έχω μία ταξινομημένη λίστα.

ΜΕΡΟΣ Δ

Στο Μέρος Δ, κάνω την **πειραματική αξιολόγηση** με διαφορετικά πλήθη φακέλων σε κάθε φάκελο.

Αρχικά κατασκευάζω τα 30 αρχεία με το κομμάτι κώδικα που φαίνεται σε σχόλια και έπειτα χρησιμοποιώντας τη **Random**, παράγω τυχαίους αριθμούς στο εύρος [0,1000000]. Αυτούς τους περνάω στα αρχεία. Τα 10 πρώτα αρχεία έχουν 100 τέτοιες τιμές (100 γραμμές), τα επόμενα 10 έχουν 500 τιμές (500 γραμμές) και τα επόμενα 10 έχουν 1000 τιμές(1000 γραμμές).

Προκειμένου να τρέξω τους 2 αλγορίθμους δημιουργώ αρχικά το αντικείμενο algorithm τύπου Greedy, μέσω του οποίου καλώ την συνάρτηση ReadFile. Αυτή θα κάνει return τη λίστα με τους φακέλους για κάθε αρχείο txt που θα διαβάσω και θα το αποθηκεύσει στη μεταβλητή folders. Χρησιμοποιώ στη συνέχεια τη λίστα folders ως όρισμα στη μέθοδο toDO, η οποία θα επιστρέψει τον αριθμό των

χρησιμοποιούμενων δίσκων στη μεταβλητή diskNoSort.

Κάνω την ίδια διαδικασία, αλλα αφού εφαρμόσω heapsort στη λίστα folders, και περνάω τον αριθμό των χρησιμοποιούμενων δίσκων στη μεταβλητή disksSort.

Έπειτα, προκειμένου να αντιληφθώ την εξοικονόμηση δίσκου που επιτυγχάνεται με την ταξινόμηση των φακέλων με βάση το μέγεθός τους, έχω τις μεταβλητές totaldisksSort και totalDisksNoSort από τις οποίες βρίσκω τον μέσο όρο των δίσκων που χρησιμοποιούνται.

	Μέσος όρος δίσκων χωρίς ταξινόμηση	Μέσος όρος δίσκων με ταξινόμηση
N=100	59	52
N=500	295	259
N=1000	585	506

Παρατηρούμε ότι με τη χρήση ταξινόμησης στα μεγέθη των φακέλων, κάνουμε πάντα εξοικονόμηση δίσκων. Φυσικά, όσο αυξάνεται ο αριθμός των φακέλων τόσο περισσότερο φαίνεται η διαφορά που επέρχεται από την ταξινόμηση.